Computing Reeb Graphs as a Union of Contour Trees

Harish Doraiswamy*
Department of Computer Science and Automation
Indian Institute of Science

Vijay Natarajan†
Department of Computer Science and Automation
Supercomputer Education and Research Centre
Indian Institute of Science

Abstract

The Reeb graph of a scalar function tracks the evolution of the topology of its level sets. This work describes a fast and efficient algorithm to compute the Reeb graph of a scalar function defined over manifolds and non-manifolds. The key idea in the proposed approach is to maximize the use of the efficient contour tree algorithm to compute the Reeb graph. The algorithm proceeds by dividing the input into a set of simply connected interval volumes using the join tree of the scalar function and computes the Reeb graph by combining the contour trees of all the interval volumes. Since the key ingredient of this method is a series of union-find operations, the algorithm is fast in practice. The algorithm also extends to handle large data that do not fit in memory.

Index Terms: I.3.5 [Computing Methodologies]: Computational Geometry and Object Modeling—Geometric algorithms, languages, and systems

1 Introduction

The Reeb graph of a scalar function is obtained by mapping each connected component of its level sets to a point. Level set components that contain critical points of the function map to nodes of the graph. The abstract representation of the level set topology in the Reeb graph facilitates the development of methods for modeling objects and visualizing scientific data. Reeb graphs and their loop-free version, called contour trees, have a variety of applications including topology-based shape matching [4] and designing transfer functions for volume rendering [9].

Rapidly increasing data sizes and the interactivity requirement in the above-mentioned applications necessitate the development of algorithms for fast computation of Reeb graphs that are capable of handling relatively large input sizes. Further, in several cases the domain is not simply connected, non-manifold, and may be high-dimensional. While an efficient and fast algorithm is available for computing contour trees in all dimensions, such an algorithm for computing Reeb graphs is still elusive. In this work we attempt to solve this problem by aggressively employing the contour tree algorithm to construct the Reeb graph. This approach results in an algorithm that is efficient both theoretically, in terms of the worst case running time, and practically, in terms of performance on real-world data. The goal of our algorithm is similar to that followed by Tierny et al. [8] – to remove loops from the input in order to use the contour tree algorithm. While their algorithm is restricted to 3-manifold input embedded in $\mathbb{R}^3$, our method is generic. The proposed algorithm is also amenable to handle large data that do not fit in memory.

2 Background

Let $\mathbb{M}$ denote a d-manifold with or without boundary. Given a smooth, real-valued function $f : \mathbb{M} \to \mathbb{R}$ defined on $\mathbb{M}$, the *critical points* of f are exactly where the gradient becomes zero. The function f is called a *Morse function* if it satisfies the following conditions [2]:

1. All critical points of f are non-degenerate and lie in the interior of $\mathbb{M}$.
2. All critical points of the restriction of f to the boundary of $\mathbb{M}$ are non-degenerate.
3. All critical values are distinct *i.e.*, $f(p) \neq f(q)$ for all critical points $p \neq q$.

The above conditions typically do not hold in practice for piecewise-linear (PL) functions. However, simulated perturbation of the function [3, Section 1.4] ensures that no two critical values are equal. A total order on the vertices helps in consistently identifying the vertex with the higher function value between a pair of vertices. In the remaining discussion, we assume that the above conditions are satisfied.

Level set topology. The preimage of a real value is called a *level set*. The level set of a regular value is a $(d-1)$-manifold with or without boundary, possibly containing multiple connected components. We are interested in the evolution of level sets against increasing function value. Topological changes occur at critical points, whereas topology of the level set is preserved across regular points [5].

The *link* of a vertex consists of all vertices adjacent to it and the induced edges, triangles, and higher-order simplices. Adjacent vertices with lower function value and their induced simplices constitute the *lower link*, whereas the adjacent vertices with higher function value and their induced simplices constitute the *upper link*. In the context of Reeb graphs, we are only interested in critical points that modify the number of level set components. So, it is sufficient to count the number of connected components of the lower and upper links for identifying these critical points. The vertex is *regular* if it has exactly one lower link component and one upper link component. All other vertices are *critical*. A critical point is a *maximum* if the upper link is empty, a *minimum* if the lower link is empty, or a *saddle* otherwise.

Reeb graphs and contour trees. The *Reeb graph* of f is obtained by contracting each connected component of a level set to a point [7]. Formally, it is the quotient space under an equivalence relation that identifies all points within a connected component of a level set. When the Reeb graph has no loops, it is called the *contour tree*.

3 The ReCon Algorithm

We now describe our Reeb graph computation algorithm (ReCon). The algorithm works without any modifications for d-manifolds, $d \geq 2$, and non-manifolds. The input to our algorithm is represented by a triangulated mesh together with a scalar function defined on the vertices of the mesh, and linearly interpolated within each simplex. The algorithm computes the Reeb graph in three stages:

1. Identify the loops present in the Reeb graph of the input and split the input at these loops to obtain a set of interval volumes.

*e-mail:harishd@csa.iisc.ernet.in
†e-mail:vijayn@csa.iisc.ernet.in

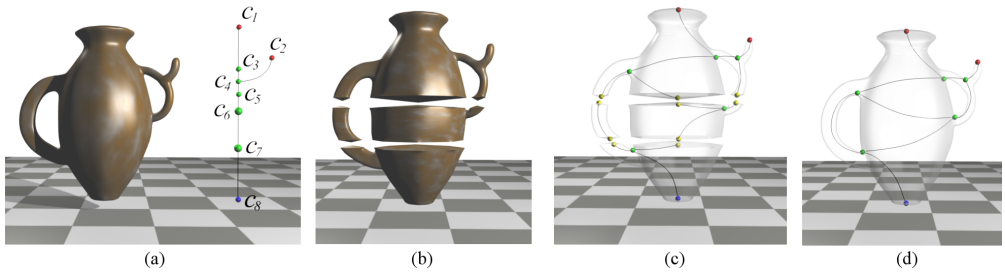

Figure 1: Computing the Reeb graph for the height function defined on a solid vase. **(a)** The join tree for the input is computed and the loop saddles c_6 and c_7 are identified. **(b)** The input is split at a function value infinitesimally above that of the loop saddles to obtain a set of interval volumes. **(c)** The contour tree for each interval volume is computed after introducing maximum-minimum pairs (yellow nodes) at each split. **(d)** The contour trees are merged at the newly created maximum-minimum pairs to obtain the Reeb graph of the input.

2. Compute the contour tree for each interval volume, which is now simply connected.
3. Construct the Reeb graph by combining the contour trees computed in the previous step.

Finding Loop saddles. Consider any loop L in the Reeb graph of the given input. If we sweep the input with decreasing function value, then a split saddle c_s *begins* the loop L, while a join saddle c_j *ends* it. We are interested in finding all such *loop saddles* – a set of saddles that begin or end a loop in the Reeb graph. The function values at these saddles are in turn used to obtain a set of loop free interval volumes. This is accomplished by splitting the input at these saddles. The following lemma provides us with the necessary condition to compute this set.

Lemma 1. *Let G_R be the Reeb graph of a scalar function f. Consider the join tree T_J of f. Any join saddle that ends a loop in G_R appears as a degree-2 node in T_J.*

Computing the Reeb graph. The algorithm first identifies all potential loops of the Reeb graph. Each loop is represented by a loop saddle (join saddle) that closes the loop. In order to locate the loop saddles, the algorithm first computes all critical points of the input scalar function f. This is accomplished by counting the number of components in the upper and lower links of every vertex. Since we are primarily interested in join saddles, we consider only those saddles whose upper link have two components. Let this set be denoted by S. The algorithm then computes the join tree T_J of f. Using Lemma 1, the algorithm creates the set of loop saddles $S_L = \{c_i \in S | deg(c_i) \text{ in } T_J = 2\}$.

Next, the algorithm implicitly splits the domain at these loops into a set of interval volumes. These interval volumes have the property that their Reeb graphs do not contain loops, and hence can be computed using the contour tree algorithm. For each saddle c_j in S_L, we perform a split in the input at a function value $f(c_j)+\varepsilon$, for an appropriate small value of ε. Each connected component that is split generates a new maximum-minimum pair. The next step computes the contour tree of the individual interval volumes.

In the final step of the algorithm, we merge the maximum-minimum pairs created in the previous step to obtain the Reeb graph of the original input. Figure 1 illustrates the algorithm for the height function defined on a solid vase model. Our algorithm has a running time of $O(n \log n + hn)$ time, where n is the number of triangles in the input and h is the number of degree-2 join saddles in the join tree.

A direct extension of our algorithm computes Reeb graphs for large data that do not fit in memory. The main idea is to split the input into interval volumes that fit into memory, and compute the Reeb graph of the input scalar function by combining the Reeb graphs of the individual interval volumes.

Table 1: Reeb graph computation time for various 2D and 3D input. The Reeb graph was computed for the height function defined by the y-axis for 2D models, while the scalar function was provided with the 3D data. *NA* denotes that the referenced algorithm is not applicable for the corresponding data set.

Model	# Triangles	Time taken (sec) ReCon	Online [6]	LS [8]
Dawn (2D)	6.6M	6.35	11.5	*NA*
Lucy (2D)	28M	43.3	60.09	*NA*
Plasma (3D)	2.6M	2.26	*NA*	1.85
Earthquake (3D)	4.2M	3.50	*NA*	2.77

4 Experimental Results

We evaluated our implementation on an Intel Xeon workstation with a 2.0 GHz processor and 16 GB main memory, see Table 1. The results show that ReCon, which is generic and is amenable to an out-of-core implementation, performs on par with or better than existing algorithms [6, 8] catered to restricted classes of input.

Acknowledgements

Harish Doraiswamy was supported by Microsoft Corporation and Microsoft Research India under the Microsoft Research India PhD Fellowship Award. This work was supported by the Department of Science and Technology, India, under Grant SR/S3/EECE/048/2007.

References

[1] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Comput. Geom. Theory Appl.*, 24(2):75–94, 2003.

[2] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Loops in Reeb graphs of 2-manifolds. *Disc. Comput. Geom.*, 32(2):231–244, 2004.

[3] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge Univ. Press, England, 2001.

[4] M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *Proc. SIGGRAPH*, pages 203–212, 2001.

[5] Y. Matsumoto. *An Introduction to Morse Theory*. Amer. Math. Soc., 2002. Translated from Japanese by K. Hudson and M. Saito.

[6] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. Robust on-line computation of Reeb graphs: simplicity and speed. *ACM Trans. Graph.*, 26(3):58, 2007.

[7] G. Reeb. Sur les points singuliers d'une forme de pfaff complètement intégrable ou d'une fonction numérique. *Comptes Rendus de L'Académie ses Séances, Paris*, 222:847–849, 1946.

[8] J. Tierny, A. Gyulassy, E. Simon, and V. Pascucci. Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees. *IEEE Trans. Vis. Comput. Graph.*, 15(6):1177–1184, 2009.

[9] G. H. Weber, S. E. Dillard, H. Carr, V. Pascucci, and B. Hamann. Topology-controlled volume rendering. *IEEE Trans. Vis. Comput. Graph.*, 13(2):330–341, 2007.